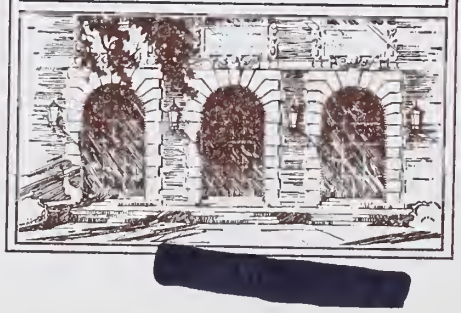

Report No. UIUCDCS-R-75-775 NSF-OCA-DCR73-07980 A02-000016

COMBINATIONAL CIRCUIT SYNTHESIS WITH TIME AND COMPONENT BOUNDS

by

S. C. Chen and D. J. Kuck

December 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-75-775

COMBINATIONAL CIRCUIT SYNTHESIS WITH TIME AND COMPONENT BONDS*

by

S. C. Chen and D. J. Kuck

December 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF DCR73-07980 A02.

1. Introduction

This paper discusses some aspects of the relationship between sequential circuits and combinational circuits. Circuit design in both areas has been studied extensively in the past. Past studies have included efforts to reduce the time and gates required to compute various functions. This paper establishes upper bounds on time and gates, and also provides a systematic procedure for transforming a sequential circuit design into a combinational circuit.

The upper bounds on time which we prove are quite good, relative to the best known lower bounds in most cases. We also give gate bounds, which have often eluded detailed analysis in the past. Our gate bounds seem quite sharp relative to the actual numbers found in real logic design examples.

The algorithms we have for transforming sequential circuit designs into combinational ones yield circuits which meet the above-mentioned gate and time bounds. In this sense, we present a uniform design procedure for the realization of any linear sequential machine in combinational circuit form. The advantage of this is that one can often specify the behavior of some desired function quite easily as a sequential circuit. It is somewhat more difficult to translate such a specification into a faster combinational circuit form. A classic example is the ease with which a bit serial adder is specified in sequential form. On the other hand, the design of combinational parallel adders (with various lookahead schemes) occupied many logic designers for some years in the 1950s. The automatic design of a fast parallel combinational adder derived from a bit serial specification is one example of the use of our method.

Not all interesting logic design problems are presented in a sequential form that is linear. As we shall see later, multiplication is an example. While some nonlinear cases can be linearized mathematically, we shall discuss another approach. We will show how nonlinear logic circuits can be used to remove the nonlinearity in the sequential specification. Then, in terms of elements which contain the nonlinearities, we obtain a linear system at a higher level. Our method can then be applied in a straightforward way.

An important question in modern, practical logic design is what to put in one integrated circuit package and then how to synthesize useful circuits using such packages. One of the methods we present deals with what can be regarded as logic design at the integrated circuit package level. We show what logic should be contained in a package and then give a method for interconnecting packages. Again our discussion is centered on transforming given sequential logic specifications into combinational logic in the form of packages. This is closely related to the subject of the previous paragraph in the sense that nonlinear logic functions can often be hidden in integrated circuit packages, leaving us with a linear problem at a higher level.

Throughout the paper we illustrate our methods with examples giving gate and time bound coefficients for several practically useful logic design problems including adders, multipliers, and ones' position counters.

The techniques described in this paper are variations on our earlier efforts to design fast parallel operation computers [1] [2].

There our basic units were adders and multipliers which operated on whole floating-point numbers, while here we are dealing with logic design at lower levels. In this paper we deal with operations on bits and bytes at the gate and integrated circuit package level. It is important to notice that mathematically, precisely the same ideas and algorithms are used at all levels; only the details of the technology change. Thus we feel that in attempts to automate the design of general purpose or special purpose machines, one set of underlying ideas may be of general use.

The following definitions and assumptions will hold throughout the paper. An atom is a constant or variable denoted by a lower case letter. In some parts of the paper we will deal with Boolean atoms (which have value 0 or 1) and in other parts we will deal with arithmetic atoms (which represent binary numbers). A dyadic Boolean operator is either a logical or or a logical and. A dyadic arithmetic operator is either an addition or multiplication operator. We denote these by + and •, respectively, in either case. The context will make our meaning clear when necessary, and in some cases the same result will hold in either the Boolean or the arithmetic case.

Except as noted in the paper, we assume that all Boolean nots and arithmetic subtractions are distributed down to the level of atoms. In the arithmetic case, this is discussed in [3], while in the Boolean case a similar procedure may be carried out using DeMorgan's Laws. We do this without loss of generality to simplify our discussion.

An expression (Boolean or arithmetic) is a well-formed string

consisting of atoms and operators and is denoted by an upper case letter. We write E<e>, for example, to denote an expression E containing e atoms. The distinction between Boolean and arithmetic atoms and expressions will be clear by the context of our discussion.

We assume throughout the paper that and, or and not gates each have one gate delay of unit time. We assume that all and and or gates have fan-in 2 and fan-out f. By dealing with such stylized gates we are able to compare various designs in elementary terms. If one assumes more complex gates with higher fan-ins, our gate and time upper bounds can obviously be reduced, in general. Another way in which the coefficients in our bounds can be uniformly improved is by ignoring the time required to complement signals. Many circuit families have gates in which both true and complemented outputs are available with no time or cost penalty. To make our bounds conservative and as widely useful as possible, we have not taken advantage of any such features.

We emphasize the fact that in practice fan-out is usually greater than fan-in, but fan-out delays may be nonnegligible. We account for fan-out delays and gates in all of our bounds. Thus our results represent a more refined treatment than is usually found in abstract bounds of this type which often ignore fan-out limitations.

We use the notation $T_G[E]$ to denote the number of gate delays in a circuit which implements expression E using G gates. Similarly, we use the notation $T_P[E]$ to denote the number of processor delays required to compute E using P processors.

Throughout the paper we use log x to denote $\log_2 x$.

2. Combinational Circuits

In this section we discuss gate and time bounds for combinational logic circuits. We give bounds for gates with fan-out f and fan-in 2. After giving some elementary fan-out and combinational fan-in bounds we present an overall circuit bound. This is expressed in terms of the number of inputs and outputs, and could, for example, be used to bound the gates and time needed for an integrated circuit package.

Throughout the paper, we assume that signals appear from some external source and are returned to some external destination after our operations on them. Effectively we are ignoring registers from which signals come and to which they are returned. Thus we can count gates and time delays and compose them in a uniform way, without ad hoc accounting procedures at the source and destination of our signals.

Our first lemma concerns the fan-out of signals and will be used extensively later.

Lemma 1 An e way fan-out can be accomplished using gates with fan-out of $f \geq 2$ in

$$T_G \leq \lceil \log_f e \rceil - 1$$

with

$$G < \frac{e-1}{f-1} .$$

Proof The first stage of fan-out is accomplished by either an external source (which we ignore) or a previous combinational gate which will be counted elsewhere. The destination of our signals is either external

(hence ignored) or combinational gates which are accounted for elsewhere. This is illustrated in Figure 1.

Thus we can fan-out to f places with zero gates, to f-1+f places with one gate, to f-2+2f places with two gates, and to f-G+Gf places with G gates. Since we want $e \leq f-G+Gf$, we see that $e \leq f-G+Gf < e+f-1$. Thus we have $G < \frac{e-1}{f-1}$.

We can fan-out to f places in zero time, to f^2 places in 1 time unit, to f^3 places in 2 time units, and to f^k places in k-1 time units. It follows that for $e \leq f^k < fe$ we have $k < 1 + \log_f e$, so $k - 1 < \log_f e$. But $k - 1 = T_G$ so the theorem is proved.

Q.E.D.

Next, we bound the gates and time in the combinational part of any logic circuit.

Lemma 2 [3], [4]

Any Boolean expression E<e> of e atoms can be realized using gates of fan-in 2 in

$$T_G[E<e>] \leq \begin{cases} 1 + 2d + \lceil \log e \rceil & \text{if } d < \frac{3}{2} \log e \\ \lceil 4 \log e \rceil & \text{otherwise,} \end{cases}$$

with

$$G[E<e>] \leq \begin{cases} e-1 & \text{if } d < \frac{3}{2} \log e \\ 2(e-1) & \text{otherwise,} \end{cases}$$

where d is the depth of parenthesis nesting in E.

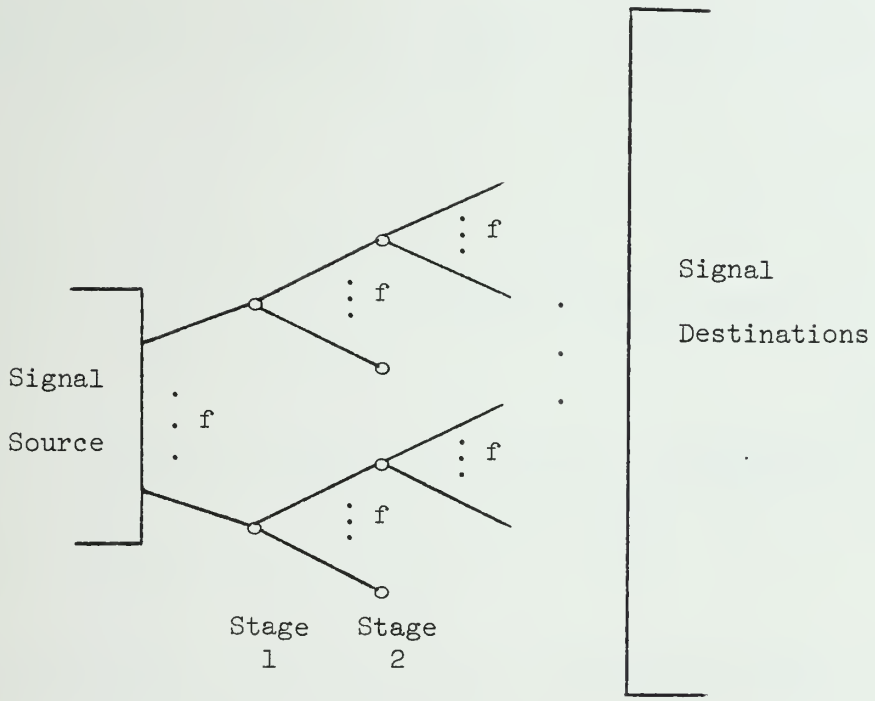

Figure 1

Signal Fan-out

The proof of this lemma for $d < \frac{3}{2} \log e$ is found in [3]. In most practical expressions, the depth of parenthesis nesting is small, so this provides the best bound. However, if $d \geq \frac{3}{2} \log e$, we use the second half of the lemma which is proved in [4], where it is also shown that this may be extended to $T_G[E<e>] \leq 3\log n$ with $G[E<e>] \leq 2.5e$. We have found that for practical purposes a low gate bound is more important than a low time bound, however. In much of the following we will use Lemma 2, assuming for simplicity that $d < \frac{3}{2} \log e$.

Next we define a combinational circuit and then give overall gate and time bounds for such circuits.

Definition 1

A combinational circuit $C<r,s,e,n,d>$ is defined by

1) A set of inputs x_i, $1 \leq i \leq r$.

2) A set of outputs y_j, $1 \leq j \leq s$, where y_j is defined by an output expression $E_j<e_j>$ of e_j atoms (representing inputs or complements of inputs) and with parenthesis nesting depth d_j.

3) $e = \max_s\{e_j\}$ is the maximum number of atoms contained in E_j, $1 \leq j \leq$

4) $n = \sum_{j=1} e_j$ is the total number of atoms in all E_j s.

5) $d = \max_j\{d_j\}$ is the maximum parenthesis nesting depth among all of the output expressions E_j.

It is clear that $n \geq s$, and we assume that $n \geq r$, i.e., each input is used in at least one output expression.

Theorem 1

Any combinational circuit $C<r,s,e,n,d>$ can be realized using gates of fan-in 2 and fan-out f in

$$T_G \leq \lceil \log e \rceil + 2(d + \lceil \log_f n \rceil)$$

with

$$G \leq (1+\frac{2}{f-1})n + (1-\frac{1}{f-1})r - s \ .$$

Proof

First, consider the fan-out of the inputs. Let the i-th input be used e_i times in output expressions. Since we may need to complement the input, we first fan it out to $e_i + 1$ places (the extra one for complementation). By Lemma 1 we need (since we assume each input atom is used at least once)

$$T_{G1} \leq \lceil \log_f (n-s+1) \rceil - 1 < \lceil \log_f n \rceil - 1$$

with

$$G1 \leq \sum_{i=1}^{r} \frac{e_i}{f-1} = \frac{n}{f-1} \ .$$

Now we can complement each input variable in

$$T_{G2} = 1$$

with $G2 \leq r$,

and fan the complemented variable out, each to at most e_i places. Thus we have

$$T_{G3} \leq \lceil \log_f (n-s+1) \rceil - 1 < \lceil \log_f n \rceil - 1$$

$$G3 \leq \sum_{i=1}^{r} \frac{e_i - 1}{f-1} = \frac{n-r}{f-1} \ .$$

Next, we consider the fan-in of the atoms to form the output variables according to the output expressions $E_j\langle e_j\rangle$. By Lemma 2 we have (assuming $d_j < \frac{3}{2}\log e_j$)

$$T_{G4} = \begin{cases} 1 + 2d + \lceil \log e \rceil & \text{if } d_j < \frac{3}{2}\log e_j,\ 1 \le j \le s \\ \lceil 4\log e \rceil & \text{otherwise} \end{cases}$$

with

$$G4 \le \begin{cases} \sum_{j=1}^{s} e_j - 1 = n - s & \text{if } d_j < \frac{3}{2}\log e_j,\ 1 \le j \le s \\ \sum_{j=1}^{s} 2(e_j - 1) = 2(n-s) & \text{otherwise.} \end{cases}$$

Thus (assuming $d_j < \frac{3}{2}\log e_j$, $1 \le j \le s$) we have a total of

$$T_G \le \lceil \log e \rceil + 2(d + \lceil \log_f n \rceil)$$

with

$$G \le \frac{n}{f-1} + r + \frac{n-r}{f-1} + n - s$$

$$= (1+\frac{2}{f-1})n + (1-\frac{1}{f-1})r - s.$$

Q.E.D.

Example 1

Suppose we have a 16 pin integrated circuit package which contains only combinational logic. Assume we can use 7 pins for inputs and 7 pins for outputs, i.e., $r = s = 7$. Assume that we have an average of 4 atoms per output expression so $n = 4.7 = 28$, the maximum number of atoms per expression is $e = 8$, and $d = 2$. Thus a typical output expression may be of the form

$$y_j = (x_1 + \bar{x}_2) * (\bar{x}_3 + x_5) .$$

Let us use circuits with fan-in 2 and fan-out 8. Now for any possible combinational logic with the above characteristics, a package can be designed such that the total package time in gate delays is

$$T_G \leq \lceil \log e \rceil + 2(d + \lceil \log_f n \rceil)$$

$$= \lceil \log 8 \rceil + 2(2 + \lceil \log_8 28 \rceil) = 3 + 2(4) = 11 .$$

The total number of gates in any such package is at most

$$G \leq (1+\tfrac{2}{7})28 + (1-\tfrac{1}{7})7 - 7 = 35 .$$

Example 2

Suppose we have a 48 pin package for large-scale integrated circuits. Let $r = s = 23$, $n = 6 \cdot 23 = 138$, $e = 16$, $d = 3$, and $f = 8$. Now any possible combinational circuit can be realized with

$$T_G \leq \lceil \log 16 \rceil + 2(3 + \lceil \log_8 138 \rceil) = 4 + 2(6) = 16$$

and

$$G \leq (1+\tfrac{2}{7})138 + (1-\tfrac{1}{7})23 - 23 = 177 .$$

Thus we see that for realistic assumptions about packages and logical expressions, we obtain gate and time bounds that are of practical interest.

3. Sequential Circuits

In this section we discuss methods of transforming sequential circuits into combinational ones and give time bounds and component bounds on the resulting circuits.

Definition 2

A sequential circuit $S<r,s,e,n,d,m>$ is defined at time t by

1) A set of inputs $x_i(t)$, $1 \leq i \leq r = r_1 + r_2$. We call the $x_i(t)$, $1 \leq i \leq r_1$, the external inputs, and the $x_i(t)$, $r_1 + 1 \leq i \leq r$, the feedback inputs.

2) A set of outputs $y_j(t)$, $1 \leq j \leq s = s_1 + s_2$, where for any logical functions f_j,

$$y_j(t) = f_j[x_1(t), x_2(t), \ldots, x_r(t)]$$

$$= f_j[x_1(t), \ldots, x_{r_1}(t), y_{s_1+1}(t-m_1), \ldots, y_s(t-m_{r_2})]$$

as shown in Figure 2 . We call the $y_j(t)$, $1 \leq j \leq s_1$, the external outputs and the $y_j(t)$, $s_1 + 1 \leq j \leq s$, the feedback outputs. Note that $r_2 \geq s_2$. Each output is defined by an output expression $E_j<e_j>$ of e_j atoms (representing inputs or their complements). Expression E_j has parenthesis nesting depth d_j.

3) $e = \{e_1, e_2\}$ where

$$e_1 = \max_{1 \leq j \leq s_1} \{e_j\} \quad \text{and} \quad e_2 = \max_{s_1+1 \leq j \leq s} \{e_j\}$$

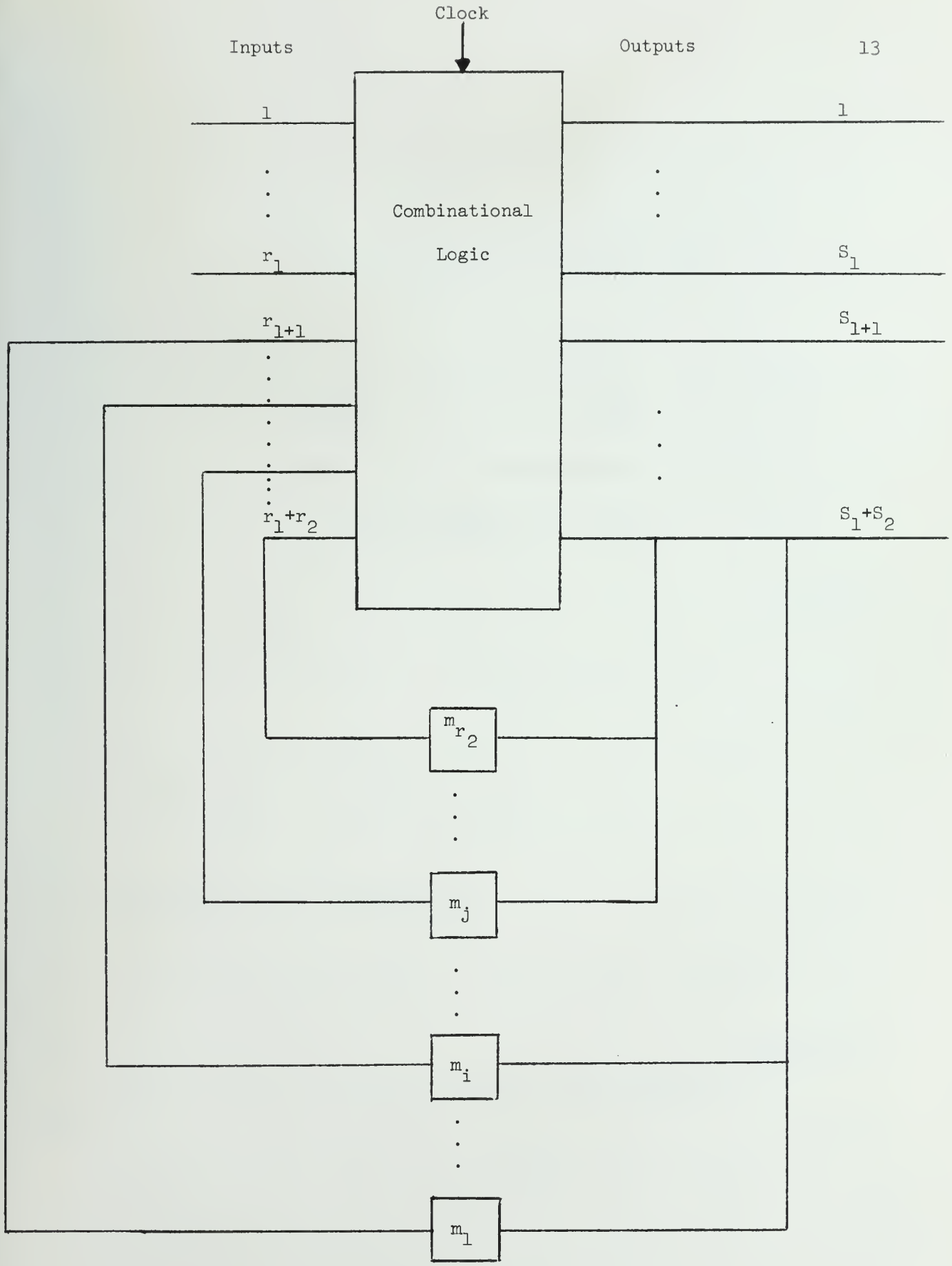

Fig. 2

Sequential Circuit

4) $n = n_1 + n_2$ where

$$n_1 = \sum_{j=1}^{s_1} e_j \quad \text{and} \quad n_2 = \sum_{j=s_1+1}^{s} e_j .$$

5) $d = \{d_1, d_2\}$ where

$$d_1 = \max_{1 \le j \le s_1} \{d_j\} \quad \text{and} \quad d_2 = \max_{s_1+1 \le j \le s} \{d_j\} .$$

6) A set of <u>delays</u> m_i, $1 \le i \le r_2$, where

$m = \max_i m_i$ is the <u>maximum delay</u>.

<u>Definition 3</u>

A <u>linear sequential circuit</u> is a sequential circuit with outputs $y_i(t)$, $s_1 + 1 \le i \le s$, of the form

$$y_i(t) = f_i[x_1(t), \ldots, x_{r_1}(t), y_{s_1+1}(t-m_1), \ldots, y_s(t-m_{r_2})]$$

$$= c_i + a_{i1}\, y_{s_1+1}(t-m_1) + \ldots + a_{ik_{r_2}}\, y_s(t-m_{r_2}) ,$$

where the c_i and a_{ij}, $1 \le j \le r_2$, are derived from any logical functions of the inputs $x_1(t), \ldots, x_{r_1}(t)$.

<u>Definition 4</u>

An <u>m-th order linear recurrence system</u> of n equations R<n,m>, is defined by

$$x_i = 0 \qquad \text{for } i \le 0,$$

and

$$x_i = c_i + \sum_{j=i-m}^{i-1} a_{ij}\, x_j \qquad \text{for } 1 \le i \le n ,$$

where $1 \leq m < n$, and the c_i and a_{ij} are constants. We assume that n and m are powers of 2. If either is not, we choose the next higher power of 2 and apply our bounds and algorithms directly. The solution of this recurrence is the set $\{x_i | 1 \leq i \leq n\}$.

The following lemma forms the basis of much of our subsequent work. We will use it to count gates as well as higher level components such as integrated circuit packages or whole processors. Thus we state the lemma in terms of operations θ which can be interpreted as logical or and and or as arithmetic addition and multiplication. When we deal with fan-out, at the gate level θ corresponds to gates while at the processor level it refers to registers or demultiplexors.

Lemma 3

Any m-th order linear recurrence R<n,m> can be solved in

$$T_\theta \leq (\frac{5}{2} + \log m + \frac{1}{2}\log_f n)\log n - \frac{1}{2}(\log^2 m + \log m)$$

with

$$\theta < \frac{1}{2}\left[m^2(2+\frac{1}{f-1}) + m(1+\frac{1}{f-1})\right]n\log n$$

$$+ \left[m^3(1+\frac{1}{f-1}) - m^2(2+\frac{1}{(f-1)}) - \frac{m}{(f-1)}\right] n + 2m^2 + (\frac{2}{f-1}) \log n$$

where $f = 2^q$, $q \geq 1$.

Proof

Our proof follows the proof of Theorem 2 of [2] and a logical circuit can be constructed following Algorithm 2 of [2]. First, we consider the time required. The computational θ delays follow directly from the time bound for solving an R<n,m> system in Theorem 2 of [2]. Thus, for the first part of our time bound, we have from Theorem 2 of [2]

$$T1 \leq (2+\log m)\log n - \frac{1}{2}(\log^2 m + \log m) .$$

To complete the time bound, we must consider the fan-out time required by Theorem 2 of [2]. Such times were regarded as negligible compared to arithmetic operation times in [2]. The solution of an R<n,m> system is generated in log n iterations. It may be seen from Figure 4 of [2] that on iteration i = log k, we perform at most $(\frac{k}{2} + m - 1)$ way fan-outs. Thus the fan-out time on iteration i is $[\log_f(\frac{k}{2} + m - 1)] - 1$, by Lemma 1. Summing over all iterations, for k = 2, 4, 8, ..., n, we have (since $f = 2^q \geq 2$),

$$T2 \leq ([\log_f 2f] - 1) + ([\log_f 4f] - 1) + \ldots + ([\log_f(\tfrac{n}{2} + m - 1)] - 1)$$

and grouping terms, we get

$$\leq q\left(1 + 2 + 3 + \ldots + \left\lceil \frac{\log n - \log 2f + 1}{q} \right\rceil\right)$$

$$= q(1 + 2 + 3 + \ldots + \lceil \log_f n \rceil - 1)$$

$$\leq q(1 + 2 + 3 + \ldots + \log_f n) = \frac{q}{2}\log_f n(1 + \log_f n)$$

$$= \frac{1}{2}\log n(1 + \log_f n) .$$

Thus our total time is T1 + T2 or

$$T_\theta \leq (2 + \log m)\log n - \frac{1}{2}(\log^2 m + \log m) + \frac{1}{2}\log n(1 + \log_f n)$$

$$= (\frac{5}{2} + \log m + \frac{1}{2}\log_f n)\log n - \frac{1}{2}(\log^2 m + \log m) .$$

Next, we consider the number of θ operations required. In the proof of Theorem 2 [2], we gave expressions for counting the number of processors in evaluating an R<n,m> system. Since a tree of n leaves has at most 2n - 1

nodes, we can upper bound the number of θ operations by doubling the processor count from Theorem 2 of [2]. We choose the worst expression for the processor count on iteration i = log k, namely, expression (2) [2], the $2m \leq 2^i \leq n$ case, sum over all iterations, for $k \in K = \{2,4,8,\ldots,n\}$, and multiply by 2 to bound the θ operations. Thus, ignoring fan-out for the moment, we have a total of

$$\theta 1 \leq 2\sum_{k\in K} \{\left[1 + (\frac{n}{k} - 2)(m + 1)\right]\left[\sum_{j=1}^{m} j + m(\frac{k}{2} - 1)\right] + (m + 1)\left[\sum_{j=1}^{m} j + m(\frac{k}{2} - m)\right]\} ,$$

where

$$K = \{2,4,8,\ldots,n\} .$$

By rearranging terms, we have

$$= 2\sum_{k\in K} \{\left[1 + (\frac{n}{k} - 1)(m + 1)\right]\sum_{j=1}^{m} j + (\frac{n}{k} - 2)\, m(m + 1)(\frac{k}{2} - 1) + m(\frac{k}{2} - 1) + m(m + 1)(\frac{k}{2} - m)\}.$$

Now summing on j gives

$$= 2\sum_{k\in K} \{[\frac{n}{k}(m+1) - m]\, \frac{m(m+1)}{2} + (\frac{n}{k} - 2)\, m(m + 1)(\frac{k}{2} - 1) + \frac{mk}{2}(m + 2) - (m^3 + m^2 + m)\}$$

$$= 2\sum_{k\in K} \{-\frac{m^2 k}{2} + \frac{(m^3 - m)n}{2}\frac{1}{k} + \frac{(m^2 + m)}{2}n - \frac{3m^3 - m^2 - 2m}{2}\}$$

$$< [-m^2(2n - 2) + (m^3 - m)n + (m^2 + m)n\log n]$$

$$= (m^2 + m)n\log n + (m^3 - 2m^2 - m)n + 2m^2 .$$

As is discussed in [2], the trees we are evaluating are of a special form with • operations at the leaf nodes and + operations elsewhere. The above sum can be used as an exact count of • operations. But since the trees are somewhat sparse, a more refined count reduces the number of + operations. Thus our factor of 2 above is too large. By a straightforward but long argument similar to the above, we can show that the θ operation count is actually bounded by

$$\theta 1 \leq (m^2 + \frac{m}{2})n \log n + (m^3 - 2m^2)n + m(2m - 1)$$

which we use in the statement of the theorem.

Now we consider the number of fan-out θ operations required. It follows from Theorem 2 [2] that iteration i requires $(m^2 + m)n/k - m^2$ fan-outs, each fanning out to at most $k/2 + m - 1$ destinations. Thus the total number of θ operations can be computed using Lemma 1 as

$$\theta 2 \leq \frac{(m^2+m)n}{f-1} \sum_{k\epsilon K} \frac{1}{k}(\frac{k}{2} + m - 2) - \frac{m^2}{f-1} \sum_{k\epsilon K} (\frac{k}{2} + m - 2) \quad , \quad K = \{2,4,8,\ldots,n\} .$$

Summing, we obtain

$$\theta 2 < \frac{(m^2+m)n}{f-1} \left[\frac{\log n}{2} + m - 1 \right] - \frac{m^2}{f-1} \left[\frac{2n-2}{2} + (m-2) \log n \right]$$

$$= \frac{m^2+m}{2(f-1)} n \log n + \frac{m^3-m^2-m}{f-1} n - \frac{m^3 \log n - 2m^2 \log n - m^2}{f-1}$$

$$< \frac{m^2+m}{2(f-1)} n \log n + \frac{m^3-m^2-m}{f-1} n + \frac{2}{f-1} \log n$$

Note that at the gate level these θ operations are gates and are comparable to the gates counted in θ1. At the integrated circuit or processor level, these θ operations correspond to registers or demultiplexors which are

generally less costly than the θ operations of $\theta 1$. But to be conservative we count each of them as one θ operation. Thus our total θ operation count is $\theta = \theta 1 + \theta 2$, so

$$\theta < \left[m^2 + \frac{m}{2} + \frac{m^2+m}{2(f-1)} \right] n \log n$$

$$+\left[m^3 - 2m^2 + \frac{m^3-m^2-m}{(f-1)} \right] n + 2m^2 + \left(\frac{2}{f-1}\right) \log n$$

$$= \frac{1}{2}\left[m^2\left(2 + \frac{1}{f-1}\right) + m\left(1 + \frac{1}{f-1}\right)\right] n \log n$$

$$+\left[m^3\left(1 + \frac{1}{f-1}\right) - m^2\left(2 + \frac{1}{(f-1)}\right) - \frac{m}{(f-1)} \right] n + \left(\frac{2}{f-1}\right) \log n + 2m^2 \quad .$$

Q.E.D.

The following corollary follows directly from Lemma 3 and covers a case of wide practical interest.

Corollary 1 Any first order linear recurrence R<n,1> can be solved in

$$T_\theta \leq \frac{1}{2}(5 + \log_f n) \log n$$

with

$$\theta \leq \frac{1}{2}\left(3 + \frac{2}{f-1}\right) n \log n - \left(1 + \frac{1}{f-1}\right) n + \left(\frac{2}{f-1}\right) \log n + 2 \quad .$$

Thus we see that for large fan-outs, we can solve any R<n,1> system in $T_G = O(\log n)$ with $G = O(n \log n)$.

Example 3

The R<8,1> system

$$c_i = 0 \ , \quad i \leq 0$$

and

$$c_i = y_i + x_i \cdot c_{i-1} \ , \quad 1 \leq i \leq 8$$

can be used to describe the carry generation in a binary adder (c.f., Theorem 3). A circuit to generate the c_i follows directly from Lemma 3 and Algorithm 2 of

of [2] by interpreting · as and, and + as or. The circuit is shown in Figure 3, assuming f = 5.

Next we give a corollary of Lemma 3 which shows the ranges of time and gates for an R<n,m> system as fan-out ranges from 2 to an arbitrarily high number.

Corollary 2 Any m-th order linear recurrence R<n,m> can be solved in

$$(2 + \log m)\log n - \frac{1}{2}(\log^2 m + \log m) \leq T_\theta \leq (\frac{5}{2} + \log m + \frac{1}{2}\log n)\log n - \frac{1}{2}(\log^2 m + \log$$

with

$$(m^2+\frac{m}{2})n \log n + (m^3-2m^2)n + O(m^2 \log n) \leq \theta \leq \frac{1}{2}\left[3m^2+2m\right]n \log n + \left[2m^3-3m^2- m\right] n + 2\log n + 2m^2$$

Proof The lower bounds follow directly from $T_{\theta 1}$ and $\theta 1$ in the proof of Lemma 3, assuming that fan-out time and θ count are negligible. The upper bounds follow from Lemma 3 by setting f = 2.

Thus we see that for large fan-outs we can solve an R<n,m> system in $T_G = O(\log m \log n)$ with $G = O(m^2 n \log n)$.

Definition 5

The k step operation of a sequential circuit S is defined by k pairs of vectors

$$[(x_1(t), \ldots, x_{r_1}(t)), (y_1(t), \ldots, y_s(t)]$$

for $1 \leq t \leq k$. These vectors represent the external inputs and outputs of S at each time step t.

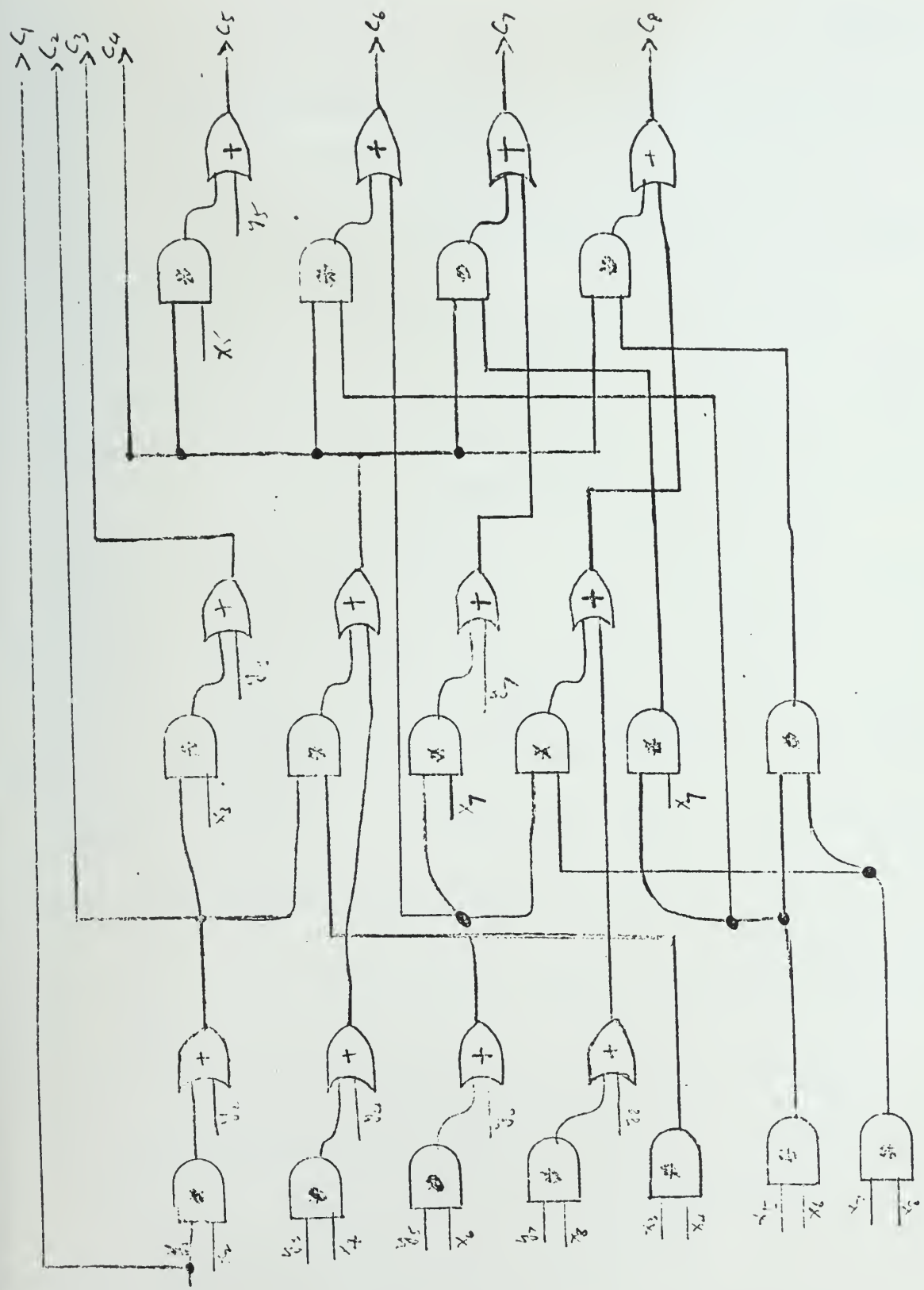

Fig. 3

An R<8,1> circuit for carry generation

Theorem 2

The k step operation of any linear sequential circuit S<r,s,e,n,d,m> can be realized by a combinational circuit such that for large k

$$T_G \leq \frac{1}{2}(\log_f s_2 k)(\log s_2 k) + O(\log k)$$

with

$$G \leq \frac{2}{3}(m+1)^2 s_2{}^3(2 + \frac{1}{f-1}) k \log s_2 k + O(k) .$$

Proof Our proof is in three parts. First, we set up the A and b arrays of Definition 4. Then we evaluate the resulting recurrence system. Finally, we generate the external outputs.

The A matrix and b vector components can be generated from the external inputs at any of the k time steps. Thus we have a total of kr_1 inputs to combinational circuit C_1 which produces as outputs the components of A and b. Since a total of n_2 atoms are used in generating all of the feedback outputs of S, there are at most kn_2 non-zero components in A and b. The maximum number of atoms in any expression is e_2, the total number of atoms is kn_2 and the maximum parenthesis depth is d_2, so we can set up the A and b arrays with

$$C_1\langle kr_1, kn_2, e_2, kn_2, d_2\rangle .$$

Next we solve the linear recurrence R<n,m> . There are a total of ks_2 outputs in k time steps so $n = ks_2$. Since the maximum delay is m time steps with s_2 outputs per time step, the bandwidth of this system is at most $(m+1)s_2 - 1$. Thus we have a recurrence of the form $R\langle ks_2, (m+1)s_2-1\rangle$.

Finally, we generate the external outputs with combinational circuit C_2. There are a total of kr inputs and ks_1 external outputs. The maximum

number of atoms in any output expression is e_1, the total number of atoms in all output expressions is kn_1 and the maximum depth is d_1, so we have

$$C_2 < kr, ks_1, e_1, kn_1, d_1 > .$$

Now we bound the gates and time required for each of these. By Theorem 1, for C_1 we have

$$T_{G1} \leq \lceil \log e_2 \rceil + 2(d_2 + \lceil \log_f kn_2 \rceil)$$

with

$$G1 \leq (1 + \frac{2}{f-1})kn_2 + (1 - \frac{1}{f-1})kr_1 - kn_2$$

$$= (1 - \frac{1}{f-1})kr_1 + \frac{2}{f-1} kn_2 .$$

By Lemma 3, we can solve $R < ks_2, (m+1)s_2 - 1 >$ in

$$T_{G2} < (\frac{5}{2} + \log(m+1)s_2 + \frac{1}{2} \log_f (m+1)s_2)\log ks_2$$

with

$$G2 < \frac{1}{2} \left[(m+1)^2 s_2^2 (2 + \frac{1}{f-1}) + (m+1)s_2(1+\frac{1}{f-1}) \right] ks_2 \log ks_2$$

$$+ (1 + \frac{1}{f-1}) (m+1)^3 s_2^3 ks_2 + O((m+1)^2 s_2^2 \log ks_2) .$$

By Theorem 1 we have for C_2

$$T_{G3} \leq \lceil \log e_1 \rceil + 2(d_1 + \lceil \log_f kn_1 \rceil)$$

with

$$G3 \leq (1 + \frac{2}{f-1}) kn_1 + (1 - \frac{1}{f-1}) kr - ks_1 .$$

Combining the above we have a total time of

$$T_G \leq \lceil \log e_1 \rceil + \lceil \log e_2 \rceil + 2(d_1 + d_2 + \lceil \log_f kn_1 \rceil + \lceil \log_f kn_2 \rceil)$$

$$+ (\frac{5}{2} + \log(m+1)s_2 + \frac{1}{2}\log_f s_2 k)\log s_2 k \ .$$

Thus, for a fixed circuit, as we increase the number of operating time steps k, we have

$$T_G \leq \frac{1}{2}(\log_f s_2 k)(\log s_2 k) + O(\log k) \ .$$

The total gate count is

$$G < \frac{1}{2}\left[(m+1)^2 s_2^2(2 + \frac{1}{f-1}) + (m+1)s_2(1 + \frac{1}{f-1})\right] s_2 k \log s_2 k$$

$$+ \left[(1 - \frac{1}{f-1})(r_1 + r) + n_1 + \frac{2}{f-1} n - s_1\right]k$$

$$+ (1 + \frac{1}{f-1})(m+1)^3 s_2^4 k + O((n+1)^2 s_2^2 \log k s_2) \ .$$

Thus, for any fixed circuit, as k increases we have

$$G \leq \frac{1}{2}\left[(m+1)^2 s_2^2(2+\frac{1}{f-1}) + (m+1)\, s_2(1+\frac{1}{f-1})\right] s_2 \, k\log k \, s_2 + O(k)$$

or (since $m \geq 1$ and $f \geq 2$)

$$G \leq \frac{2}{3}(m+1)^2 s_2^3(2+\frac{1}{f-1})\, k\log s_2 k + O(k) \ .$$

Q.E.D.

Now we turn to the consideration of higher level components as our basic circuit elements. We will define two package types which could be implemented directly using integrated circuits. Our time bounds will be expressed in package delays. The techniques of the previous section could be used to design such packages. Our component bounds will be expressed in terms of the total number of packages required.

Our strategy in this case is to decompose a linear recurrence system R<n,m> into a number of small identical systems. These smaller systems can be solved directly by interconnecting the integrated circuit packages we specify. An algorithm to decompose a large R<n,m> system has been given in [2], [5] for arithmetic operations. Here we present the algorithm for logic design and consider only the R<n,1> case for the sake of easy explanation. The R<n,1> case is by far the most common one occurring in practical logic design, and our method can be extended to larger m in a straightforward way.

Definition 6

We define two types of integrated circuit packages.

a) $IC_{R<n,1>}$ is a package which accepts input atoms c_i for $1 \leq i \leq n$, and a_i for $2 \leq i \leq n$. It computes the outputs x_i for $1 \leq i \leq n$ according to the recurrence relation

$$x_0 = 0$$

$$x_i = c_i + a_i x_{i-1} .$$

For signal input and output it has a total number of pins equal to $3n - 1$ times the number of bits per atom.

b) $IC_{U<n>}$ is a package which may accept input atoms a_i and b_i for $1 \leq i \leq n$, and c and d. It computes the outputs x_i for $1 \leq i \leq n$, according to

$$x_i = v_i w_i + y_i z_i ,$$

where either

i) $v_i = a_i$, $w_i = c$, $y_i = b_i$ and $z_i = d$, $1 \leq i \leq n$

or

$$\text{ii)}\quad v_i = a_i,\ w_i = b_i,\ y_i = \overline{a}_i \quad \text{and} \quad z_i = \overline{b}_i,\ 1 \leq i \leq n.$$

For signal input and output it has a total number of pins of at most 3n + 2 times the number of bits per atom. In general, we denote the total number of integrated circuits in some logical circuit by IC.

Example 4

An $IC_{R<4,1>}$ has a total of 3·4-1 = 11 signal pins if it is to solve a Boolean recurrence. Suppose we are summing the bits in a 16-bit word and will produce a log 16 = 4 bit result. Then 4 bits are required per atom and an arithmetic $IC_{R<4,1>}$ to solve this problem would need 44 signal pins. An $IC_{U<3>}$ for Boolean operations requires 3·3+2 = 11 signal pins. An arithmetic package for handling 4 bit numbers would need a total of 44 signal pins.

The following algorithm is adapted from [5] (c.f. Ch. 4). It solves any R<n,1> system by partitioning it into smaller systems.

Algorithm 1 Any given first-order linear recurrence

$$R<n,1>:\quad x_0 = 0$$

$$x_i = c_i + a_i\, x_{i-1},\ 1 \leq i \leq n$$

can be solved as follows.

Step 1

a) For any $h \geq 2$, compute $\frac{n}{h}$ independent recurrence systems $Z^{(j)}$, $1 \leq j \leq \frac{n}{h}$, defined as follows.

$$Z^{(j)}: \quad z_0^{(j)} = 0 \ ,$$

$$z_i^{(j)} = c_i^{(j)} + a_i^{(j)} \, z_{i-1}^{(j)} \ , \quad 1 \leq i \leq \frac{n}{h} \ ,$$

where

$$c_i^{(j)} = c_{i+(j-1)h} \ ,$$

$$a_i^{(j)} = a_{i+(j-1)h} \ .$$

b) Compute $(\frac{n}{h}-1)$ independent recurrence systems $Y^{(j)}$, $2 \leq j \leq \frac{n}{h}$, defined as follows.

$$Y^{(j)}: \quad y_0^{(j)} = 1 \ ,$$

$$y_i^{(j)} = a_i^{(j)} \, y_{i-1}^{(j)}, \quad 1 \leq i \leq h \ .$$

From this step we obtain h elements of the solution of the original system, i.e., $x_i = z_i^{(1)}$ for $1 \leq i \leq h$.

Step 2

From the results of Step 1, compute the following recurrence system

$$\underset{\sim}{z}_h^{(0)} = 0$$

$$\underset{\sim}{z}_h^{(j)} = z_h^{(j)} + y_k^{(j)} \, z_h^{(j-1)} \ , \quad 1 \leq j \leq \frac{n}{h} \ .$$

From this step we obtain another $(\frac{n}{h}-1)$ elements of the solution, i.e., $x_{jh} = \underset{\sim}{z}_h^{(j)}$ for $2 \leq j \leq \frac{n}{h}$.

Step 3

From the results of Steps 1 and 2, compute the remaining elements

of the solution using the following $n - \frac{n}{h} - (h-1)$ independent expressions:

$$x_{i+(j-1)h} = z_i^{(j)} + y_i^{(j)} \underset{\sim}{z}_h^{(j-1)} \quad \text{for } 1 \leq i \leq h-1 \text{ and } 2 \leq j \leq \frac{n}{h} .$$

Lemma 4 Any first-order linear recurrence R<n,1> can be solved in time

$$T_{IC} \leq (2 \frac{\log n}{\log h} - 1)$$

using a total package count of

$$IC \leq 6\frac{n}{h} + 4 \frac{\log n}{\log h} - 7$$

with package types $IC_{R<h,1>}$ and $IC_{U<h-1>}$ for $h \geq 2$.

Proof

It follows directly from the above algorithm that we need one $IC_{R<h,1>}$ type package for each $Z^{(j)}$ and $Y^{(j)}$ in Step 1a and 1b. This results in $(2\lceil \frac{n}{h} \rceil - 1)$ packages. In Step 3 we use $(\lceil \frac{n}{h} \rceil - 1)$ packages of type $IC_{U<h-1>}$, corresponding to Definition 6b, part i. We can treat Step 2 as a new $R<\lceil \frac{n}{h} \rceil, 1>$ system and apply the same algorithm recursively to solve this system. This implies that we reduce the size of the original system from n to less than or equal to h, following the sequence $n' = n, \lceil \frac{n}{h} \rceil, \lceil \lceil \frac{n}{h} \rceil \frac{1}{h} \rceil, \ldots, h$, and finally use one extra $IC_{R<h,1>}$ package to solve the residual system. Hence, for each iteration we need $(2\lceil \frac{n'}{h} \rceil - 1)$ packages of type $IC_{R<h,1>}$ and $(\lceil \frac{n}{h} \rceil - 1)$ packages of type $IC_{U<h-1>}$. Since at most $\frac{\log n}{\log h} - 1$ iterations are required, we have for $IC_{R<h,1>}$ type package a total of $IC = (2\lceil \frac{n}{h} \rceil - 1) + (2\lceil \lceil \frac{n}{h} \rceil \frac{1}{h} \rceil - 1) + \ldots$

$$\leq \left\lceil 2(\frac{n}{h}+1)-1 \right\rceil + \left\lceil 2(\frac{n}{h^2} + \frac{1}{h} + 1)-1 \right\rceil + \ldots$$

$$\leq 2\frac{n}{h}(1 + \frac{1}{h} + \frac{1}{h^2} + \ldots) + 3(\frac{\log n}{\log h} - 2) + 1 ,$$

and since $h \geq 2$,

$$\leq \frac{4n}{h} + 3\frac{\log n}{\log h} - 5 .$$

Similarly, for $IC_{U<h-1>}$ type packages, we have a total of

$$IC = (\left\lceil \frac{n}{h} \right\rceil - 1) + (\left\lceil \left\lceil \frac{n}{h} \right\rceil \frac{1}{h} \right\rceil - 1) + \ldots$$

$$\leq (\frac{n}{h}) + (\frac{n}{h^2} + \frac{1}{h}) + (\frac{n}{h^3} + \frac{1}{h^2} + \frac{1}{h}) + \ldots$$

$$\leq \frac{n}{h}(1 + \frac{1}{h} + \frac{1}{h^2} + \ldots) + (\frac{\log n}{\log h} - 1) - 1$$

$$\leq \frac{2n}{h} + \frac{\log n}{\log h} - 2 .$$

The time bound is obtained by the fact that all packages in the same step per iteration are operating in parallel. Q.E.D.

Example 5 The R<16,1> system

$$x_i = 0 \qquad \text{for } i \leq 0$$

and

$$x_i = c_i + a_i\, x_{i-1} \qquad \text{for } 1 \leq i \leq 16 ,$$

can be solved by the circuit of Figure 4 which follows directly from Algorithm 1 with $h = 4$. The packages marked R represent $IC_{R<4,1>}$ types and those marked U represent $IC_{U<3>}$.

For use in a later application, we now consider a special case of an R<n,1> system. Let $a_i = 1$, for all i, in Algorithm 1. In this case we

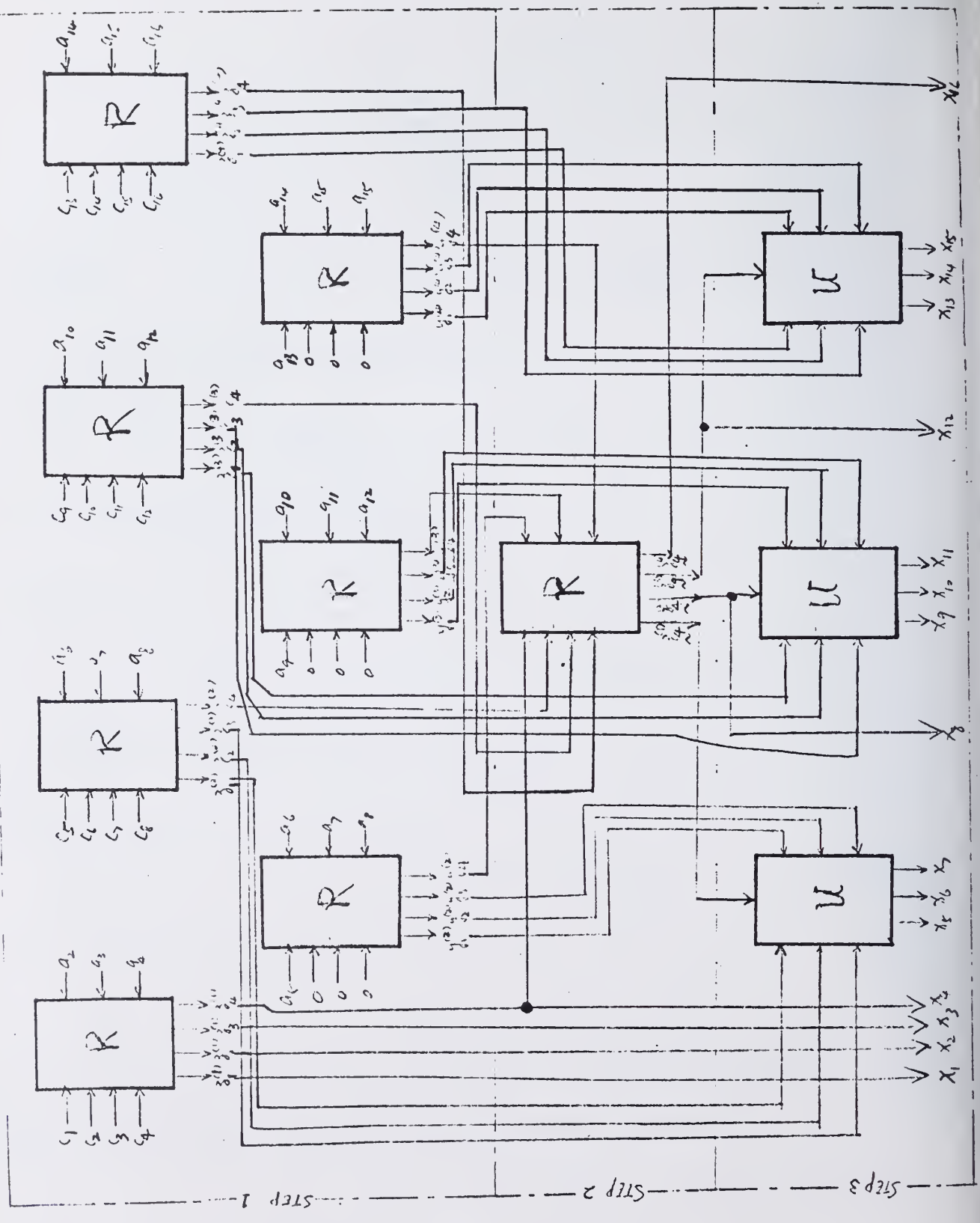

Fig. 4 Package Interconnection for R<16,1> system, with h = a

need not perform step 1b. So for each iteration, only $\left\lceil \frac{n'}{h} \right\rceil$ type $IC_{R<h,1>}$ packages are required. Also, note that all $Z^{(j)}$ are computed in Step 1a by merely summing atoms. Since Steps 2 and 3 require only multiplication by the y's generated in Step 1, which are 1's, no multiplication is required in any package. From this we have

Corollary 3 Any R<n,1> system of the form

$$x_0 = 0$$

$$x_i = c_i + x_{i-1},\ 1 \leq i \leq n$$

can be solved in time

$$T_{IC} \leq (2\frac{\log n}{\log h} - 1)$$

using a total package count of

$$IC \leq 4\frac{n}{h} + 3\frac{\log n}{\log h} - 4 .$$

4. Applications

In this section we will study several practical logic design problems. The methods of section 3 will be used to derive time and component bounds. We will consider binary addition and ones' position counting in detail. In less detail we will consider binary multiplication, digital filtering and a control problem.

Definition 7

By the addition of two n digit binary numbers $a = a_n \ldots a_1$ and $b = b_n \ldots b_1$ we mean the generation of sum digits $s = s_n \ldots s_1$ and carry digit c_n, defined as follows.

We write

$$s_i = (a_i b_i + \bar{a}_i \bar{b}_i)\, c_{i-1} + (\bar{a}_i b_i + a_i \bar{b}_i)\, \bar{c}_{i-1} \qquad (1)$$

where $1 \leq i \leq n$ and $c_0 = 0$, such that $s_i = 1$ iff just one or all three of a_i, b_i and c_{i-1} are equal to 1. Also we write

$$c_i = a_i b_i + (a_i + b_i)\, c_{i-1} \qquad (2)$$

where $1 \leq i \leq n$ and $c_0 = 0$, such that $c_i = 1$ iff any two or all three of a_i, b_i and c_{i-1} are equal to 1. Now let

$$x_i = a_i + b_i \qquad (3)$$

and

$$y_i = a_i b_i \,. \qquad (4)$$

If we write

$$d_i = a_i b_i + \bar{a}_i \bar{b}_i = \overline{(a_i + b_i)} + a_i b_i = \bar{x}_i + y_i \qquad (5)$$

then Equation 1 can be rewritten as

$$s_i = d_i\, c_{i-1} + \bar{d}_i\, \bar{c}_{i-1} \tag{6}$$

and Equation 2 can be rewritten as

$$c_i = \begin{cases} y_i + x_i\, c_{i-1} & 1 \le i \le n \\ 0 & i = 0 \,. \end{cases} \tag{7}$$

Our first result concerns binary addition using gates as components.

Theorem 3

Two $n = 2^t$, $t \ge 0$, digit binary numbers can be added in

$$T_G \le \frac{1}{2}(5+\log_f n) \log n + 4$$

with

$$G \le (\frac{3}{2} + \frac{1}{f-1})\, n \log n + (8 - \frac{1}{f-1})\, n + (\frac{2}{f-1}) \log n + 2 \,.$$

Proof Our proof consists of three parts.

1) To generate the x_i and y_i, $1 \le i \le n$, from a_i and b_i by Equations 3 and 4, we need 2n gates and one gate delay, so $T_{G1} = 1$ and $G1 = 2n$.

2) To generate the s_i, $1 \le i \le n$, from x_i, y_i, and c_{i-1} using Equation 6, we refer to Figure 5. A total of 7 gates are required for each s_i, for a total of 7n gates. After d_i and c_{i-1} are available, three gate delays are required. It will be seen in part 3 that the generation of the c_i, $1 \le i \le n$, from x_i and y_i can be accomplished in 2log n steps. So for $n \ge 2$ the two steps required to generate d_i from x_i and y_i are no more than the time required to generate c_i, since 2log 2 = 2.

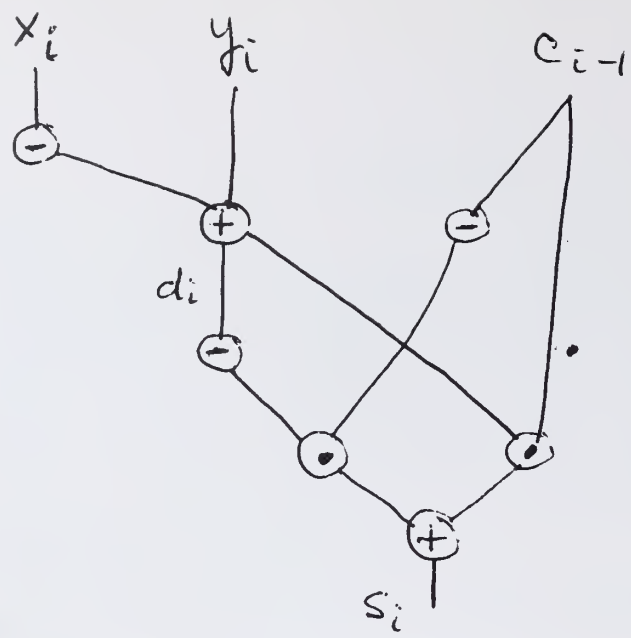

Figure 5
Sum Generation

It is easy to verify that the theorem holds for n = 1 by a direct construction. Thus we have $T_{G2} = 3$ with $G2 \leq 7n$.

3) To generate the c_i, $1 \leq i \leq n$, from x_i and y_i using Equation 7, we turn to Lemma 3. Since Equation 7 defines an R<n,1> system, it follows immediately from Corollary 1 (c.f., Figure 3) that

$$T_{G3} \leq \frac{1}{2}(5 + \log_f n) \log n$$

with

$$G3 \leq \left[\frac{3}{2} + \frac{1}{f-1}\right] n \log n - (1 + \frac{1}{f-1}) n + \frac{2}{f-1} \log n + 2 .$$

Thus we have from parts 1, 2 and 3 a total of

$$T_G = 1 + 3 + \frac{1}{2}(5 + \log_f n) \log n$$

$$= \frac{1}{2}(5 + \log_f n) \log n + 4$$

with

$$G \leq 2n + 7n + (\frac{3}{2} + \frac{1}{f-1}) n \log n - (1 + \frac{1}{f-1}) n + \frac{2}{f-1} \log n + 2$$

$$= (\frac{3}{2} + \frac{1}{f-1}) n \log n + (8 - \frac{1}{f-1}) n + (\frac{2}{f-1}) \log n + 2 .$$

Q.E.D.

Next we consider binary addition with integrated circuit packages as components.

Theorem 4

Two $n = 2^t$, $t \geq 0$, digit binary numbers can be added in time

$$T_{IC} \leq (2\frac{\log n}{\log h} + 1)$$

using a total package count of

$$IC \leq 9\frac{n}{h} + 4\frac{\log n}{\log h} - 7$$

with package types $IC_{R<h,1>}$ and $IC_{U<h>}$ for $h \geq 2$.

Proof The x_i and y_i of Definition 7 can be generated in one package delay using 2n/h type IC_U packages. The carries of Equation 7 (c.f., Figure 4) can be generated following Lemma 4 in $T_{IC} \leq (2\frac{\log n}{\log h} - 1)$ using $6\frac{n}{h} + 4\frac{\log n}{\log h} - 7$ packages. Then the sum bits of Equation 6 can be generated in one package delay using $\frac{n}{h}$ packages of type IC_U following Definition 6b, part ii. Summing these counts proves the theorem.

Q.E.D.

Example 6

Consider the problem of adding two 32-bit binary numbers using gates with fan-in 2 and fan-out 8. By the method of Theorem 3, the sum can be formed in at most 21 gate delays since

$$T_G \leq \frac{1}{2}(5 + \log_8 32) \log 32 + 4$$

$$< \frac{1}{2}(\frac{20}{3})\ 5 + 4 = \frac{100}{6} + 4 < 21 .$$

The number of gates required is at most

$$G \leq (\frac{3}{2} + \frac{1}{7})\ 32 \cdot 5 + (8 - \frac{1}{7})\ 32 + 2 \cdot 5 + 2 < \frac{23}{14} \cdot 160 + \frac{55}{7} \cdot 32 + 12 = 527$$

On the other hand, if integrated circuit packages are available which handle 8 bits at a time, h = 8, we have the following. The total package count is

$$IC \leq 9\ \frac{32}{8} + 4\ \frac{\log 32}{\log 8} - 7 < 37$$

and the number of package delays is

$$T_{IC} \leq (2\ \frac{\log 32}{\log 8} + 1) < 5\ .$$

The next application we study is a ones' position counter. This is the problem of determining the number of ones to the right (say) of each bit in a word. The problem arises in various real world contexts, particularly in control design. We discuss the problem because of its practical interest and also because it serves as an interesting case standing between binary addition and binary multiplication.

As we saw above, given the theoretical background of section 3 on solving linear recurrences, the design of a binary adder is straightforward. The ones' position counter is not as easy, however. When formulated at the bit level, this problem leads to a nonlinear recurrence which cannot be solved by the methods of section 3. As we shall see later, binary multiplication also shares this property.

The technique we use to solve such logic design problems with bit level nonlinearities, is to reformulate them at a higher level where they are in fact linear. The nonlinearity is thus hidden inside a more complex bit level operator. In practical terms, this can be accomplished by building a nonlinear circuit element and then combining these in linear ways according to the techniques of section 3. Putting such nonlinearities inside integrated

circuit packages is an attractive possibility.

Definition 8

The ones' position counting of an n bit word $a = a_n \ \ldots \ a_2 a_1$ is the generation of a count vector $z = (z_n, \ldots, z_1)$ such that z_i is the sum of the number of ones in bits $a_i \ \ldots \ a_1$. Thus, the ones' position count of a = 10110110 is the vector z = (5,4,4,3,2,2,1,0).

Following Definition 8, we can easily generate the z vector using the following arithmetic R<n,1> system

$$z_0 = 0$$

$$z_i = a_i + x_{i-1} , \quad 1 \leq i \leq n .$$

Thus by using log n bit adders (c.f., Theorem 4) as components we can solve the system in O(log n) adder steps (c.f., Corollary 1), so

$$T_G = O(\log n)\ O(\log \log n) = O(\log n \log \log n) .$$

Since each adder has O(log n log log n) gates, we have a total gate count of

$$G = ((n \log n) \cdot O(\log n \log \log n)$$
$$= O(n \log^2 n \log \log n) .$$

By formulating this problem in terms of integrated circuit packages we can use Corollary 3 to achieve a better gate count than the above. Thus, to solve an arithmetic R<n,1> system we need $IC = O(\frac{n}{h})$. Each $IC_{R<n,1>}$ package is used to count 1's, so inside each package we can use the method of Corollary 1 to solve an arithmetic R<h,1> system. Thus from Corollary 1, we have $\theta = O(h \log h)$. Now let us choose h = log n so $\theta = O(\log n \log \log n)$. Each such θ processor is used to add log n bit numbers. Thus we use Theorem 3 to count the gates as G = O(log n log log n). Multiplying these

three levels of components we obtain a total gate count of

$$G = O(\frac{n}{h} \cdot h \log h \cdot \log n \cdot \log \log n) = O(n \log n \cdot (\log \log n)^2) .$$

Similarly, we obtain the time. By Corollary 3 we have $O(\log n/\log h)$ package delays. Each package delay is $T_\theta = O(\log h)$ from Corollary 1. And the add time by Theorem 3 is $O(\log \log n)$. Hence, our total time in gate delays is

$$T_G = O(\log n \cdot \log \log n) .$$

Thus we see that the time is the same but we have reduced the gate count over the straightforward method. We can summarize this as

Theorem 5

The ones' position count of an $n = 2^t$, $t \geq 0$, bit word can be generated in

$$T_G = O(\log n \cdot \log \log n)$$

with

$$G = O(n \cdot \log n \cdot (\log \log n)^2) .$$

We note that the gate count can be further improved by using more types of packages. For example, if we let $h = \log \log n$ in Step 1 of Algorithm 1 and $h = \log n$ in Step 2 (see proof of Lemma 4), we can obtain a solution in

$$T_G = O(\log n \cdot \log \log n)$$

with

$$G = O(n \cdot (\log \log n)^2 (\log \log \log n)) .$$

By using even more package types, even better gate bounds are possible.

To obtain a package bound, Corollary 3 can be applied directly. The following example illustrates this.

Example 7

Suppose we have packages of types $IC_{R<4,1>}$ and $IC_{U<3>}$ as illustrated

in Example 4. Then by Corollary 3, the ones' position count of an n-vector, with n = 16, can be done with at most $2(\frac{16}{4}) + 2(\frac{4}{2}) - 2 = 10$ $IC_{R<4,1>}$ packages and $2(\frac{16}{4}) + (\frac{4}{2}) - 2 = 8$ $IC_{U<3>}$ packages. Actually, by direct application of Algorithm 1, it can be easily found that we need just five $IC_{R<4,1>}$ packages and three $IC_{U<3>}$ packages. The following table shows the package count for some practical values of n.

package \ n		16	32	64
Bound	$IC_{R<4,1>}$	10	19	36
	$IC_{U<3>}$	8	17	33
Actual	$IC_{R<4,1>}$	5	11	21
	$IC_{U<3>}$	3	8	18

Table 1. Ones' Position Count

Finally, let us consider a bit level formulation of this problem. Following Definition 8, let z_{ij} be the j-th, $1 \leq j \leq 1 + \log n$, bit of z_i. We can imagine solving the problem using an array of half adders such that the half adder in position (i,j) is described by ($\oplus$ denotes exclusive or)

$$z_{ij} = z_{i-1,j} \oplus c_{i,j-1} \qquad \text{Eq. 8}$$

$$c_{ij} = z_{i-1,j} \cdot c_{i,j-1} \qquad \text{Eq. 9}$$

where $c_{i,0} = a_i$ and $s_{0,1} = 0$. Notice that at the bit level, this is a non-linear recurrence and cannot be solved by the methods of section 3.

If we use half-adders as components, it is easy to see that the problem can be solved with $G = O(n \log n)$ or $T_G = O(n)$. This gate count is comparable to the best shown above, but it uses much more time.

Next, we turn to bounds for binary number multipliers.

Definition 9 By the multiplication of two n digit binary numbers $a = a_n \cdots a_1$ and $b = b_n \cdots b_1$, we mean the generation of 2n product digits $p = p_{2n} \cdots p_1$.

First, we can formulate the multiplication problem using a straightforward (row parallel) carry-save adder array. If we let x correspond to various pairwise ands of input bits [6], we obtain a coupled recurrence system of the form

$$q_{ij} = x \oplus q_{i-1,j} \oplus c_{i-1,j-1} \qquad \text{Eq. 10}$$

$$c_{ij} = x \cdot q_{i-1,j} + x \cdot c_{i-1,j-1} + q_{i-1,j} \cdot c_{i-1,j-1} . \qquad \text{Eq. 11}$$

Note that this nonlinear recurrence system is a generalization of Equations 8 and 9 for the ones' position counter. This cannot be solved by the methods of section 3, however, we can solve it directly using an array of n^2 bit level adders. This gives a circuit which can multiply two n bit numbers in $T_G = O(n)$ with $G = O(n^2)$. Since we are interested in faster schemes, we will now turn to two methods to solve the recurrence of Equations 10 and 11 in parallel.

The first method uses a tree of 2n bit adders. First, we form a standard array of partial products. Then we use the adder tree to form the sum.

Theorem 6

Two $n = 2^t$, $t \geq 0$, digit binary numbers can be multiplied in

$$T_G \leq \frac{1}{2}(6 + \log_f n)\log^2 n + \frac{1}{2}(14 + \log_f n)\log n + \log_f n + 1$$

with

$$G < (3 + \frac{2}{f-1})n^2 \log n + (20 + \frac{2}{f-1})n^2 - 3n \log n - (17 - \frac{2}{f-1})\, n .$$

Proof Our proof consists of two parts: 1) To generate the n^2 partial product bits $a_i \cdot b_j$, for all $1 \leq i,j \leq n$ we need n^2 and gates and one gate delay. Since each input bit is fanned out to n places we have from the above and Lemma 1,

$$T_{G1} \leq 1 + \log_f n$$

with

$$G1 \leq n^2 + 2n\left(\frac{n-1}{f-1}\right) < n^2\left(1 + \frac{2}{f-1}\right)$$

2) To generate the sum of the partial products we need an adder tree of n - 1 adders. Each adder adds 2n bit numbers and the height of the tree is log n adder delays. Thus, by Theorem 3, we have

$$T_{G2} \leq \log n \cdot \left[\frac{1}{2}(5 + \log_f 2n)\log 2n + 4\right]$$

$$= \frac{1}{2}(6 + \log_f n)\log^2 n + \frac{1}{2}(14 + \log_f n)\log n$$

with

$$G_2 = (n-1)\left[\left(\frac{3}{2} + \frac{1}{f-1}\right) 2n \log 2n + \left(8 - \frac{1}{f-1}\right) 2n + \frac{2}{f-1} \log 2n + 2\right]$$

$$< \left(3 + \frac{2}{f-1}\right) n^2 \log n + 19n^2 - 3n \log n - \left(17 - \frac{2}{f-1}\right) n .$$

Q.E.D.

As an example of this theorem, consider an integrated circuit package as follows.

Example 8

Using gates with fan-out 8, a multiplier of two 4-bit numbers can be implemented with a delay of

$$T_G \leq \frac{1}{2}(6) \log^2 4 + \frac{1}{2}(14) \log 4 + 1 = 20$$

using

$$G \leq \left(3 + \frac{2}{7}\right) 4^2 \log 4 + \left(20 + \frac{2}{7}\right) 4^2 - 3 \cdot 4 \cdot \log 4 - \left(17 - \frac{2}{7}\right) 4 = 341 .$$

The above result is somewhat sloppy because we considered all inputs to the tree adder to be 2n bit numbers. In fact, the inputs to the first level of adders are only n bit numbers. At succeeding levels they are of length $n + 2, n + 5, \ldots, n + i + 2^{i-1} - 2$, for $1 \leq i \leq \log n$. By a careful analysis which takes this increasing length into account, we can improve the gate count in Theorem 6 by a factor between 2 and 3. Thus, in our example the gate count could actually be bounded by a number between 115 and 170.

The method above is the best method we know (in terms of time) for numbers with few digits. For long numbers, the next method is the best we know. The crossover between the two occurs between 8 and 16 bits.

The next method is a variation of the Wallace-Dadda method [7], [8]. It consists of three stages; generation of partial products, column compression, and a carry propagate adder. This differs from Wallace-Dadda only in the last stage.

The generation of partial products is done in the same way as in Theorem 6. For an upper bound on time, we assume a three to two column compression scheme [6]. The column compression for two n-bit numbers can be done with $(n^2 - 4n + 3)$ full adders and $(n - 1)$ half adders. The half adder can be built using 9 gates (see Theorem 3 with $n = 1$). A full adder of 2 bits can be easily implemented with 11 gates by a scheme similar to Figure 5. Thus we have a total of

$$G2 \leq 11(n^2 - 4n + 3) + 9(n - 1) = 11n^2 - 35n + 24 .$$

The time for the column compression is

$$T_{G2} \leq 6\log_{3/2} n = 10\log n$$

since each full adder requires at most 6 gate delays.

Finally, to propagate the carry, we use a 2n bit adder (fewer bits are actually needed)

$$T_{G3} \leq (\frac{5}{2} + \frac{1}{2}\log_f 2n)$$

as we used in Theorem 6. This leads us to

Theorem 7

Two $n = 2^t$, $t \geq 0$, digit binary numbers can be multiplied in

$$T_G \leq (13 + \frac{1}{2}\log_f n) \log n + \frac{3}{2}\log_f n + 8$$

with

$$G \leq (12 + \frac{2}{f-1}) n^2 + (3 + \frac{2}{f-1}) n \log n - 16n + \frac{2}{f-1} \log n + (26 + \frac{2}{f-1}) .$$

Example 9

Using gates of fan-in 8, we can multiply two 32-bit numbers in

$$T_G \leq (13 + \frac{1}{2}\log_8 32) \log 32 + \frac{3}{2}\log_8 32 + 8 < 81$$

with

$$G \leq (12 + \frac{2}{7}) 1024 + (3 + \frac{2}{7}) 32\log 32 - 512 + \frac{2}{7}\log 32 + 26 + \frac{2}{7} < 12{,}624$$

Thus far, all of our examples have dealt with R<n,1> systems. In practical logic design linear recurrence systems with $m > 1$ also arise.

First, consider the following logical path tracing problem. Suppose we are given two binary words $a = a_n \ldots a_1$ and $b = b_n \ldots b_1$ and a starting bit, either a_1 or b_1. We wish to generate a word $e = e_n \ldots e_1$ which consists of those bits on a path through a and b chosen as follows. First, we let e_1 be the given starting bit. Then we choose bits in the same word until we encounter a zero in (say) bit i, which causes us to choose bit e_{i+1} from the other word. We continue in the other word until we encounter a zero which causes another switch, etc. We define

$$c_i = a_{i-1} \cdot c_{i-1} + \bar{b}_{i-1} \cdot d_{i-1} \ , \ 2 \leq i \leq n$$

and

$$d_i = \bar{a}_{i-1} \cdot c_{i-1} + b_{i-1} \cdot d_{i-1} \ , \ 2 \leq i \leq n$$

as two control words, where $c_1 = 1$ if a_1 is our starting bit and $d_1 = 1$ if b_1 is our starting bit.

Then we have

$$e_i = a_i \cdot c_i + b_i \cdot d_i$$

The generation of $c = c_n \ldots c_1$ and $d = d_n \ldots d_1$ can be handled as a coupled linear recurrence system of the form R<2n,3> .

As a final application of the ideas of this paper we mention digital filtering. This topic has received a great deal of attention in recent years. Our combinational results can be applied to nonrecursive filters and our recurrence results can be applied to recursive filters in rather direct ways. For more details about such filters see [9] or [10].

REFERENCES

[1] R. Brent, D. Kuck, and K. Maruyama, "The Parallel Evaluation of Arithmetic Expressions Without Division," IEEE Transactions on Computers, Vol. C-22, No. 5, pp. 532-534, May 1973.

[2] S. C. Chen, and D. Kuck, "Time and Parallel Processor Bounds for Linear Recurrence Systems," IEEE Transactions on Computers, Vol. C-24, No. 7, pp. 701-717, July 1975.

[3] D. Kuck, and Y. Muraoka, "Bounds on the Parallel Evaluation of Arithmetic Expressions Using Associativity and Commutativity," Acta Informatica, Vol. 3, Fasc. 3, pp. 203-216, 1974.

[4] R. P. Brent, "The Parallel Evaluation of Arithmetic Expressions in Logarithmic Time," Complexity of Sequential and Parallel Numerical Algorithms, J. F. Traub, ed., Academic Press, N.Y., 1973.

[5] S. C. Chen, "Speedup of Iterative Programs in Multiprocessor Systems," Ph.D. thesis, Univ. of Ill. at Urb.-Champ., Dept. of Computer Science Report No. 694, Jan. 1975. (NSF - OCA -GJ-36936 - 000004).

[6] A. Habibi, and P. A. Wintz, "Fast Multipliers," IEEE Transactions on Computers, Vol. C-19, No. 2, pp. 153-57, Feb. 1970.

[7] C. S. Wallace, "A suggestion for a fast multiplier," IEEE Transactions on Electronic Computers, Vol. EC-13, pp. 14-17, Feb. 1964.

[8] L. Dadda, "Some schemes for parallel multipliers," Alta Frequenza, Vol. 31, pp. 319-356, March 1965.

[9] W. D. Little, "An Algorithm for High-Speed Digital Filters," IEEE Transactions on Computers, Vol. C-23, No. 5, pp. 466-469, May 1974.

[10] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An Approach to the Implementation of Digital Filters," IEEE Transactions on Audio and Electroacoustics, Vol. AU-16, No. 3, Sept. 1968.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-775	2.	3. Recipient's Accession No.

4. Title and Subtitle

Combinational Circuit Synthesis with Time and Component Bounds

5. Report Date: December 1975

6.

7. Author(s)

S. C. Chen and D. J. Kuck

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

US NSF DCR73-07980 A02

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered

Technical Report

14.

15. Supplementary Notes

16. Abstracts

New results are given concerning the design of combinational logic circuits. We give time and component bounds for combinational circuits specified in several ways. For any sequential machine defined by linear recurrence relations, we discuss an algorithm for the synthesis of equivalent combinational logic. The procedure includes upper bounds on the time and components involved. We also discuss the transformation of nonlinear recurrences into combinational circuits. Examples are given using gates as well as ICs as components. These include binary addition, multiplication, and ones' position counting. The time and component bounds our procedure yields compare favorably with traditional results.

17. Key Words and Document Analysis. 17a. Descriptors

Binary addition
Binary multiplication
Circuit synthesis
Combinational circuits
Component bounds
Sequential circuits
Time bounds

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement

Release Unlimited

19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 48
20. Security Class (This Page UNCLASSIFIED	22. Price